Reservation and Checkpointing Strategies for Stochastic Jobs

Ana Gainaru, Brice Goglin, Valentin Honoré, Guillaume Pallez, Padma Raghavan, Yves Robert, Hongyang Sun

Reservation and Checkpointing Strategies for Stochastic Jobs

A. Gainaru, B. Goglin, V. Honoré, G. Pallez (Aupy), P. Raghavan, Y. Robert, and H. Sun

ROMA team, Inria Grenoble Rhône-Alpes, France
LIP laboratory, ENS Lyon, France
STORM team, Inria Bordeaux Sud-Ouest, France
LaBRI laboratory, Bordeaux, France
Vanderbilt University, Nashville, TN, USA
Innovative Computing Laboratory, Knoxville, TN, USA

Abstract

In this paper, we are interested in scheduling and checkpointing stochastic jobs on a reservation-based platform, whose cost depends both (i) on the reservation made, and (ii) on the actual execution time of the job. Stochastic jobs are jobs whose execution time cannot be determined easily. They arise from the heterogeneous, dynamic and data-intensive requirements of new emerging fields such as neuroscience. In this study, we assume that jobs can be interrupted at any time to take a checkpoint, and that job execution times follow a known probability distribution. Based on past experience, the user has to determine a sequence of fixed-length reservation requests, and to decide whether the state of the execution should be checkpointed at the end of each request. The objective is to minimize the expected cost of a successful execution of the jobs. We provide an optimal strategy for discrete probability distributions of job execution times, and we design fully polynomial-time approximation strategies for continuous distributions with bounded support. These strategies are then experimentally evaluated and compared to standard approaches such as periodic-length reservations and simple checkpointing strategies (either checkpoint all reservations, or none). The impact of an imprecise knowledge of checkpoint and restart costs is also assessed experimentally.

1 Introduction

In this paper, we revisit our recent work on reservation strategies for stochastic jobs [3]. Stochastic jobs originate from Big Data or Machine Learning workloads, whose performance is widely dependent on characteristics of input data. Figure 1 shows an example of a Neuroscience job. Reservation strategies provide a sequence of fixed length reservations to execute a stochastic job. If the reservation is too short for the job, it is restarted in a longer reservation. We extend the approach to include the possibility of checkpointing at the end of some (well-chosen) reservations. The idea of checkpointing is very natural and widely used in practice, in particular for long jobs lasting several hours, but it dramatically complicates the design of scheduling strategies. To the best of our knowledge, existing approaches either checkpoint at the end of all reservations,

or never. For large-scale applications, checkpointing to save intermediate results at the end of each reservation is the de facto standard approach.

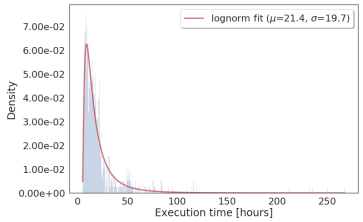


Figure 1: Execution times from 2017 for a *Structural identification of orbital anatomy* application, and its fitted distribution (in red).

We use an example to help understand the challenges of the problem under study. Consider the jobs depicted in Figure 1. We model their execution time with $\mathcal{D}$, a truncated LogNormal probability distribution on the domain $[a, b] = [0, 80\text{h}]$ (mean $\mu = 21\text{h}$, standard deviation $\sigma = 20\text{h}$). The exact execution time X of the next job to be scheduled is not known until that job has successfully completed, but instead is randomly and uniformly sampled from the target probability distribution $\mathcal{D}$. We want to minimize the expected cost of scheduling this job. To do so, we have to derive a sequence of reservations. Then we compute the cost of the job given that sequence, and aim at minimizing the expected value. To determine the cost of a reservation, we use the generic model from our previous work [3]. This model has been shown to encompass a variety of scenarios, ranging from the *Reserved Instances* of Cloud Computing where one pays (for a cheaper cost) only the reserved time [2], to High-Performance Computing (HPC) platforms where one pays the total execution time (wait time and runtime).

Specifically, for a reservation of length W_1 and an actual execution duration of length X, the cost is expressed as:

$$\alpha W_1 + \beta \min(W_1, X) + \gamma \tag{1}$$

where α, β and γ are constant parameters that depend on the platform and the cost model. The first component αW_1 is proportional to the reservation length (pay for what you ask). The second component $\beta \min(W_1, X)$ is proportional to the actual execution time (pay for what you use). Finally, the third and last component is a start-up time possibly associated with the first and/or second components.

To illustrate the contribution of this work, we use $\alpha = 1, \beta = \gamma = 0$ in the example, and we divide execution costs by a factor 60 for simplicity, so that $[a, b] = [0, 80]$ minutes. In Figure 2 , we depict three strategies, and their expected costs: (i) S_1 (Standard), which reserves the upper bound of $\mathcal{D}$, $W_1 = b = 80$; (ii) S_2 (No Checkpoint), which introduces a first reservation of size $W_1 = 20$ before the second reservation $W_2 = 80$; (iii) S_3 (With Checkpoint), which introduces a first checkpointed reservation of size $W_1 = 20 + 7$ (20 to

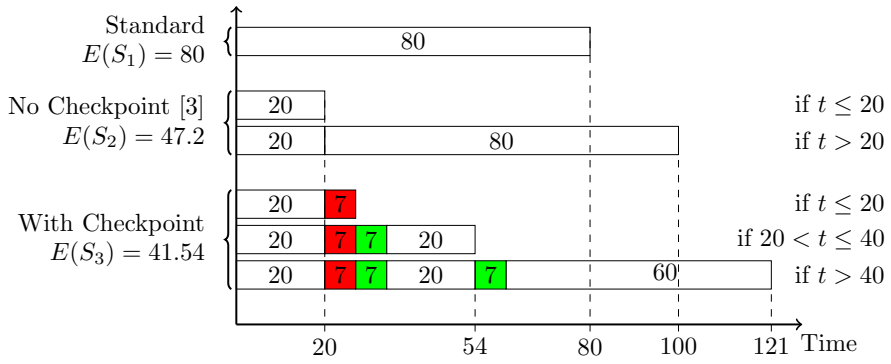


Figure 2: Illustration of different reservation strategies. The checkpoint (red) and restart (green) costs are equal to 7.

cover jobs shorter than 20, and 7 (red box) is the cost to checkpoint), then a second non-checkpointed reservation of size $W_2 = 7 + 20$ (7 (green box) is the cost to restart, 20 to cover jobs larger than 20 and smaller than 40), and a third reservation of size $W_3 = 7 + 60$ (7 is the cost to restart, 60 to cover jobs of size up to b). Here are the expected costs of these strategies:

$$\begin{aligned}
\mathbb{E}(S_1) &= 80 \\
\mathbb{E}(S_2) &= 20 \cdot \mathbb{P}(X \leq 20) + 100 \cdot \mathbb{P}(20 < X \leq 80) \\
&= 20 \times 0.66 + 100 \times 0.34 = 47.2 \\
\mathbb{E}(S_3) &= 27 \cdot \mathbb{P}(X \leq 20) + 54 \cdot \mathbb{P}(20 < X \leq 40) + 121 \cdot \mathbb{P}(40 < X) \\
&= 27 \times 0.66 + 54 \times 0.26 + 121 \times 0.08 = 41.54
\end{aligned}$$

Note that $\tilde{S}_3$, the variant of S_3 where the second reservation is also checkpointed, would have a larger expected cost due to this second checkpoint: $\mathbb{E}(\tilde{S}_3) = 27 \times 0.66 + 61 \times 0.26 + 128 \times 0.08 = 43.92$. Similarly one can verify that not performing the second reservation at all would have also increased the expected cost. This example shows that checkpointing does help for some scenarios but has too much overhead for others, and suggests that finding the best trade-off is difficult.

Indeed, in the general case, one has to decide which reservations should be checkpointed, depending on application profile and platform parameters. Moreover, determining the expected cost of a given reservation sequence together with scheduling decisions gets quite complicated. Section 2 gives a detailed formula for the expected cost, and Theorem 1 in Section 3.1 provides a simplified version. In our previous work without checkpoints [3], we have been able to analytically characterize the optimal sequence of reservations for any smooth probability distribution (except the length of the first reservation which had to be found numerically). The problem with checkpoints is dramatically more difficult, but we provide a holistic approach: we show how to compute the optimal solution for any discrete probability distribution, using a sophisticated dynamic programming algorithm. Then we show how to approximate the opti-

mal solution for any continuous probability distribution with bounded support, by providing a reservation sequence (and its checkpointing decisions) whose expected cost is arbitrarily close to the optimal one. In practice, the restriction to bounded support is not a limitation. Given, say, a LogNormal or Weibull probability distribution defined on $[a, \infty)$, it is very natural to truncate it on a bounded interval $[a, b]$ where b corresponds to the quantile $Q(1-\varepsilon)$ for a small value of ε. This amount to discarding job execution times that are unreasonably too long, and never encountered in practice.

The main contributions of this work are the following:

- The characterization of an optimal reservation sequence, together with its checkpointing decisions, for any discrete probability distribution, using a sophisticated dynamic programming algorithm.
- An approximation of the optimal solution for any continuous probability distribution with bounded support, by providing an algorithm to compute a reservation sequence (and its checkpointing decisions) whose expected cost is arbitrarily close to the optimal one.
- An extensive set of simulation results as well as experiments on a multicore platform, using nine probability distributions and neuroscience application traces, showing the efficiency of our strategies in a HPC environment.

The rest of the paper is organized as follows. Section 2 introduces the framework and main notations, and provides a detailed formula for the expected cost of a reservation sequence and its checkpointing decisions. Section 3 describes our key algorithmic contributions. Section 4 is devoted to experimental evaluation and comparison with existing approaches. Section 5 evaluates the actual performance of the approach for a neuroscience application on an HPC platform. Section 6 presents related work. Finally, we provide concluding remarks and hints for future work in Section 7.

2 Framework

In this section, we introduce some notations and formally define the optimization problem under study.

2.1 Stochastic jobs

We consider stochastic jobs whose execution times are unknown but (i) deterministic with respect to input data, so that two successive executions of the same job will have the same duration; and (ii) randomly and uniformly sampled from a given probability distribution law $\mathcal{D}$, whose density function (PDF) is f and cumulative distribution function (CDF) is F. The probability distribution is assumed to be nonnegative, since we model execution times, and it is defined either on a finite support $[a, b]$, where $0 \leq a < b$, or on an infinite support $[a, \infty)$ where $a \geq 0$. Hence, the execution time of a job is a random variable X, and $\mathbb{P}(X \leq T) = F(T) = \int_a^T f(t)dt$. For notational convenience, we sometimes extend the domain of f outside the support of $\mathcal{D}$ by letting $f(t) = 0$ for

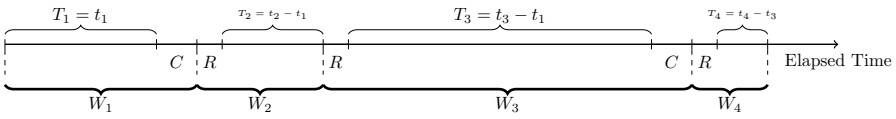


Figure 3: Illustration of the elapsed time for the reservation sequence $\mathcal{S} = \{(W_1, 1), (W_2, 0), (W_3, 1), (W_4, 0)\}$.

$t \in [0, a] \cup [b, \infty)$.

In addition, we assume that we can interrupt the jobs at any time (divisible load application) to take a checkpoint: this will save the current progress of the execution, and enable to restart from that point on. Divisible load applications can be found, for example, in biological computations, image and video processing [20]. We assume that the cost of checkpoint and of recovery is constant throughout the execution: let C be the cost to checkpoint the data at the end of an execution, and R the cost to read the data to restart a computation.

2.2 Cost model

We use the cost model motivated in our previous work [3]. For a reservation of length W and an actual execution duration w for the job, the cost is $\alpha W + \beta \min(W, w) + \gamma$, where $\alpha > 0$, $\beta \geq 0$ and $\gamma \geq 0$. If the job does not complete within W seconds, then another reservation should be paid for.

However, we take checkpoints into account in this work. If the job did not complete its execution during the last reservation, but was checkpointed during the last C seconds of that reservation, then in the current reservation, the job can restart from that checkpoint during the first R seconds, and then continue execution from its saved state. On the contrary, if no checkpoint was taken during the last reservation, the work done during that reservation is lost, and the execution must restart from the last checkpoint (or from the very beginning if no checkpoint was taken yet).

Altogether, the user needs to schedule a (possibly infinite) sequence of reservations $\mathcal{W} = (W_1, W_2, \dots, W_i, W_{i+1}, \dots)$ to execute any job whose execution time follows the distribution $\mathcal{D}$, and to launch these reservations one after the other, until the job successfully terminates within the duration of some reservation. In addition, the user should decide whether to take a checkpoint or not at the end of each reservation.

Definition 1 (Reservation sequence for $\mathcal{D}$)**.** Given a probability distribution $\mathcal{D}$, a *reservation sequence* $\mathcal{S} = \{(W_1, \delta_1), (W_2, \delta_2), \dots\}$, is defined as a sequence of reservation lengths W_k and a sequence of checkpointing decisions $\delta_k \in \{0, 1\}$: $\delta_k = 1$ means the k^{th} reservation ends with a checkpoint, and $\delta_k = 0$ means it does not.

Then, the k^{th} reservation can be decomposed into:

$$W_k = R_k + T_k + C_k \tag{2}$$

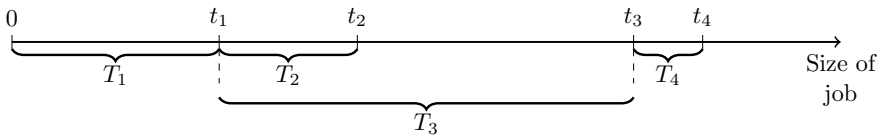


Figure 4: Illustration of job progress (and showing t_k versus T_k) for the reservation sequence $\mathcal{S} = \{(W_1, 1), (W_2, 0), (W_3, 1), (W_4, 0)\}$.

where R_k is the time spent for restart, T_k for actual job execution, and C_k for checkpoint. We have $C_k = \delta_k C$ by definition. There is a restart if and only if there has been a checkpoint at some point before, hence $R_k = (1 - \prod_{i=1}^{k-1}(1 - \delta_i))R$ (assuming $R_1 = 0$ for the first reservation). But it is hard to keep track of actual job progress when using only the (W_k, δ_k) values. Consider for instance the following sequence $\mathcal{S} = \{(W_1, 1), (W_2, 0), (W_3, 1), (W_4, 0)\}$, which is depicted in Figure 3. If the actual job duration is $X = t$, during which reservation will the job complete its execution? We introduce another view of the reservation sequence $\mathcal{S}$ by introducing the milestones $\{t_k\}$ as shown in Figure 4. A milestone t_k represents the amount of work that has been actually executed at the end of the k^{th} reservation. Then, the last reservation for the job of length t is W_k, where $t_{k-1} \leq t \leq t_k$. Of course, we need that $t \leq t_4$ for all values of $\mathcal{D}$ (equivalently, the upper bound of the support of $\mathcal{D}$ is $b \leq t_4$) for all jobs to complete successfully with the four reservations of $\mathcal{S}$.

The relationship between the milestone t_k (actual work progress) and the value of T_k (time spent computing during reservation W_k; see Equation (2)) is $t_k = T_k + \sum_{i=1}^{k-1} \delta_i T_i$: Indeed, the work actually progresses only from the last checkpoint, while the work executed during the previous non-checkpointed reservations is lost whenever these non-checkpointed reservations do not allow for the full completion of the job. Another way to express the relationship between t_k and T_k is the following:

$$t_k = T_k + \max\{t_i \mid 1 \leq i \leq k-1 \text{ and } \delta_i = 1\} \tag{3}$$

Equation (3) gives a recursive way to compute t_k from its definition. We recapitulate the relations between all notations introduced in Figures 3 and 4:

$$\begin{array}{ll} W_k = R_k + T_k + C_k; & R_k = (1 - \Pi_{i<k}(1 - \delta_i))R; \\ T_k = t_k - \sum_{i<k} \delta_i T_i; & C_k = \delta_k C. \end{array} \tag{4}$$

In the following, we use milestones t_k rather than reservation lengths W_k to characterize a reservation sequence, and we write $\mathcal{S} = \{(t_1, \delta_1), (t_2, \delta_2), \dots\}$ instead of $\mathcal{S} = \{(W_1, \delta_1), (W_2, \delta_2), \dots\}$, because it is easier to use milestones when computing the expected cost of a sequence, as shown below. For notational convenience, we define $t_0 = 0$ as the first milestone of each sequence $\mathcal{S}$. Note also that we can restrict to sequences where $t_{k-1} < t_k$, because otherwise (if $t_{k-1} = t_k$), the execution does not progress during the k^{th} reservation.

2.3 Expected cost

Given a reservation sequence $\mathcal{S} = ((t_i, \delta_i))_i$ and a job with execution time t such that $t_{k-1} < t \leq t_k$, the cost of the sequence for that job is given by:

$$\begin{aligned} C_{\mathcal{S}}(k,t) = &\sum_{i=1}^{k-1} (\alpha W_i + \beta W_i + \gamma) \\ &+ \alpha W_k + \beta(R_k + t - (t_k - T_k)) + \gamma \end{aligned} \tag{5}$$

where the first part is the total cost from the $k-1$ first reservations that did not allow the job to complete, and the second part is the cost of the k^{th} reservation. The actual execution time during the k^{th} reservation is $t - (t_k - T_k)$, because $t_k - T_k$ is the amount of work done up to the beginning of that reservation; we add the restart time (R_k) but do not need to checkpoint (if $\delta_k = 1$) because the job successfully completes before it is taken.

We let $k(t) = k$ for a job of length t such that $t_{k-1} < t \leq t_k$. Now, given a random variable X following a distribution $\mathcal{D}$, the expected cost of the reservation sequence $\mathcal{S}$ is

$$\mathbb{E}(\mathcal{S}(X)) = \int_0^\infty C_{\mathcal{S}}(k(t), t) f(t) dt = \sum_{k=1}^{\infty} \int_{t_{k-1}}^{t_k} C_{\mathcal{S}}(k, t) f(t) dt \tag{6}$$

2.4 Optimization problem

We are now ready to state the optimization problem:

Definition 2 (Stochastic)**.** Given a random variable X (with PDF f and CDF F) for the execution times of a stochastic job, and a cost function in Equation (5) (with parameters $\alpha > 0$ and $\beta, \gamma \geq 0$), find a reservation strategy $\mathcal{S}$ with minimal expected cost $\mathbb{E}(\mathcal{S}(X))$ as given in Equation (6).

We further define ReservationOnly to be the instance of Stochastic where the cost is a linear function of the reservation length only, i.e., when $\beta = \gamma = 0$. For ReservationOnly, we can further consider $\alpha = 1$ without loss of generality. For instance, such costs are incurred when making reservations of resources to schedule jobs on some cloud platforms, with hourly or daily rates. Throughout the paper, we focus on the usual probability distributions, hence we assume that the density function f and the CDF F of $\mathcal{D}$ are smooth (infinitely differentiable), and that $\mathcal{D}$ has finite expectation.

3 Algorithms

In this section, we establish some key properties of an optimal solution in the general setting.

3.1 Expected cost

We start by establishing a simpler expression for the expected cost function of STOCHASTIC in the following theorem. The proof is omitted due to space constraint but can be found in the companion report [9].

Theorem 1. *Given a random variable (RV) X and a reservation sequence $\mathcal{S} = ((t_1, \delta_1), (t_2, \delta_2), \dots)$, the expected cost $\mathbb{E}(\mathcal{S}(X))$ of a strategy $\mathcal{S}$ given by Equation (6), with parameters α, β and γ, can be rewritten as*

$$\begin{aligned}\mathbb{E}(\mathcal{S}(X)) = \beta \cdot \mathbb{E}[X] + \alpha\big(t_1 + \delta_1 C\big) + \gamma + \sum_{i=2}^{\infty} \Big(\alpha W_i \\ + \beta\big(R_i + (1-\delta_{i-1})T_{i-1} + C_{i-1}\big) + \gamma\Big) \cdot \mathbb{P}(X > t_{i-1})\end{aligned} \tag{7}$$

For ease of reading, when there is no ambiguity on the RV X, we write $\mathbb{E}(\mathcal{S}(X)) = \mathbb{E}(\mathcal{S})$.

3.2 Dynamic programming for discrete distributions

We study the problem for a finite discrete distribution: $Y \sim (v_i, f_i)_{1 \le i \le n}$, where $v_i < v_{i+1}$ for all $1 \le i \le n-1$ and $f_i = \mathbb{P}(Y = v_i)$. We assume that $f_n \neq 0$ and $\sum_{i=1}^{n} f_i = 1$. Consider a strategy $\mathcal{S} = \{(t_1, \delta_1), (t_2, \delta_2), \dots, (t_{|\mathcal{S}|}, \delta_{|\mathcal{S}|})\}$, where $t_i = v_{\pi(i)}$ and $t_i < t_{i+1}$ for all $1 \le i \le |\mathcal{S}| - 1$. Also, the last reservation is necessarily $t_{|\mathcal{S}|} = v_n$ to ensure that the expected cost of the strategy is finite. By convention, we let $t_0 = v_0 = a$, hence $\mathbb{P}(Y > t_0) = 1$. Note that we can safely restrict to strategies where each milestone t_i is equal to some threshold v_j of the discrete distribution: otherwise, replacing t_i by the largest v_j such that $v_j \le t_i$ leads to a smaller cost.

Rewriting Equation (7) with $W_i = R_i + T_i + C_i$, and since $W_0 = 0$, the expected cost of strategy $\mathcal{S}$ can be expressed as:

$$\begin{aligned}\mathbb{E}(\mathcal{S}) = \beta \cdot \mathbb{E}[Y] + \sum_{i=1}^{|\mathcal{S}|} \big(\alpha\,(R_i + T_i + C_i) + \beta R_i + \gamma\big) \cdot \mathbb{P}(Y > t_{i-1}) \\ + \sum_{i=1}^{|\mathcal{S}|-1} \beta\big((1-\delta_i)T_i + C_i\big) \cdot \mathbb{P}(Y > t_i)\end{aligned} \tag{8}$$

Based on Equation (8), and using Equations (4), we construct a dynamic programming algorithm to compute the optimal reservation sequence:

Theorem 2. *For a discrete distribution $Y \sim (v_i, f_i)_{1 \le i \le n}$, the optimal expected*

cost is returned by $\mathbb{E}_{\text{ckpt}}(0,0)$*, where, for* $0 \le i_c \le i_l \le n$*,* $\mathbb{E}_{\text{ckpt}}(i_c, i_l)$ *is:*

$$
\begin{aligned}
&= \beta \cdot \mathbb{E}[Y], && \text{if } i_l = n \\
&= \min_{\substack{i_l+1 \le j \le n, \\ \Delta_j \in \{0,1\}}} \Bigg(\mathbb{E}_{\text{ckpt}}\big(\Delta_j j, j\big) + \Big(\alpha\big(v_j + \Delta_j C\big) + \gamma\Big) \cdot \sum_{k=i_l+1}^{n} f_k \\
&\quad + \beta\big((1-\Delta_j)v_j + \Delta_j C\big) \cdot \sum_{k=j+1}^{n} f_k \Bigg), && \text{if } i_c = 0 \\
&= \min_{\substack{i_l+1 \le j \le n, \\ \Delta_j \in \{0,1\}}} \Bigg(\mathbb{E}_{\text{ckpt}}\big((1-\Delta_j) i_c + \Delta_j j, j\big) \\
&\quad + \Big(\alpha\big(R + (v_j - v_{i_c}) + \Delta_j C\big) + \beta R + \gamma\Big) \cdot \sum_{k=i_l+1}^{n} f_k \\
&\quad + \beta\big((1-\Delta_j)(v_j - v_{i_c}) + \Delta_j C\big) \cdot \sum_{k=j+1}^{n} f_k \Bigg), && \text{otherwise}
\end{aligned}
$$

The optimal solution can be computed in $O(n^3)$ *time.*

Intuitively, i_c denotes the index of the last checkpointed value, while i_l denotes the index of the last value that was tried before we try the next one with index j. Here, Δ_j indicates whether the value v_j will be checkpointed or not. The optimality is proven by induction on the index of the last checkpointed reservation. The proof is again omitted but available in the companion report [9].

3.3 Approximation algorithm for continuous distributions

In this section, we provide an approximation algorithm of the optimal strategy for continuous distributions with bounded support $[a, b]$, where $a \ge 0$ and b is finite. Because we model job execution times, it is natural to truncate continuous distributions whose support is $[0, \infty[$ such as an Exponential or LogNormal distribution, say, to a bounded support $[a, b]$.

The result for continuous distribution is particularly important: we have shown in recent work [10] that continuous distributions gave strategies that allowed using small data samples to find an efficient strategy. Here, it returns an arbitrarily good quality solution with low complexity.

More precisely, let X be a continuous random variable defined on $[a, b]$ modeling the probability distribution $\mathcal{D}$, where $0 \le a < b$, with CDF F and PDF f. Theorem 3 shows that Algorithm 1 computes a close-to-optimal strategy for STOCHASTIC. Before stating Theorem 3, we start with a lemma:

Lemma 1. *Given a random variable* X *and a strategy* $\mathcal{S} = \{(t_1, \delta_1), \ldots, (t_{|S|}, \delta_{|S|})\}$*, if there exists an index* $i_0 > 1$ *such that* $t_1 < \cdots < t_{i_0-1} \le \min(R, \varepsilon\mathbb{E}[X]) < t_{i_0} < \cdots < t_{|S|}$*, then the strategy* $\tilde{\mathcal{S}} = \{(\min(R, \varepsilon\mathbb{E}[X]), 0), (t_{i_0}, \delta_{i_0}), \ldots, (t_{|S|}, \delta_{|S|})\}$ *satisfies:*

$$\mathbb{E}(\tilde{\mathcal{S}}(X)) \le (1 + \varepsilon) \cdot \mathbb{E}(\mathcal{S}(X))$$

Algorithm 1 DYN-PROG-COUNT(X, ε)

Let $[a, b]$ be the domain of X, with $0 \leq a < b$
$c_0 = 3(b-a)\min\left(\frac{1}{\min(\max(a, \varepsilon\mathbb{E}[X]/3), R, C)}, \frac{\alpha+\beta}{\gamma}\right)$
$n \leftarrow \lceil c_0/\varepsilon \rceil$
Define the discrete distribution $Y_n \sim (v_i, f_i)_{i=1\ldots n}$ s.t.

$$\begin{cases} v_i & = a + i \cdot \frac{b-a}{n} & \text{for } 0 \leq i \leq n \\ f_i & = \mathbb{P}(Y_n = v_i) = \mathbb{P}(v_{i-1} < X \leq v_i) & \text{for } 1 \leq i \leq n \end{cases} \tag{9}$$

$\mathcal{S}_n^{\mathrm{dp}} \leftarrow$ Optimal strategy for Y_n (Theorem 2)
return $\mathcal{S}_n^{\mathrm{dp}}$

Intuitively, this lemma states that restricting to strategies whose first reservation length is at least $\min(R, \varepsilon\mathbb{E}[X])$, increases the cost by at most a factor of $1 + \varepsilon$.

Proof. Consider a strategy $\mathcal{S} = \{(t_1, \delta_1), \ldots, (t_{|S|}, \delta_{|S|})\}$ for a random variable X, such that there exists an index $i_0 > 1$ with $t_1 < \cdots < t_{i_0-1} \leq \min(R, \varepsilon\mathbb{E}[X]) < t_{i_0} < \cdots < t_{|S|}$. Let $\tilde{a} = \min(R, \varepsilon\mathbb{E}[X])$, and define strategy $\tilde{\mathcal{S}} = \{(\tilde{a}, 0), (t_{i_0}, \delta_{i_0}), \ldots, (t_{|S|}, \delta_{|S|})\}$. From Equation (7), we have $\mathbb{E}(\mathcal{S}(X)) \geq \beta\mathbb{E}[X] + \alpha(t_1 + C_1) + \gamma +$
$(\alpha W_{i_0} + \beta\big(R_{i_0} + (1-\delta_{i_0-1})T_{i_0-1} + C_{i_0-1}\big) + \gamma\Big) \cdot \mathbb{P}(X > t_{i_0-1}) + \sum_{i=i_0+1}^{|\mathcal{S}|} \Big(\alpha W_i +$
$\beta\big(R_i + (1-\delta_{i-1})T_{i-1} + C_{i-1}\big) + \gamma\Big) \cdot \mathbb{P}(X > t_{i-1})$, while
$\mathbb{E}(\tilde{\mathcal{S}}(X)) = \beta\mathbb{E}[X] + (\alpha\tilde{a} + \gamma) + (\alpha\tilde{W}_{i_0} + \beta\tilde{a} + \gamma) \cdot \mathbb{P}(X > \tilde{a}) + \sum_{i=i_0+1}^{|\mathcal{S}|} \Big(\alpha\tilde{W}_i + \beta\big(\tilde{R}_i +$
$(1-\delta_{i-1})\tilde{T}_{i-1} + \tilde{C}_{i-1}\big) + \gamma\Big) \cdot \mathbb{P}(X > t_{i-1})$.

We obviously have $C_i = \tilde{C}_i,\ \forall i \geq i_0$. We now show that $\forall i \geq i_0,\ W_i \geq \tilde{W}_i$. We consider two cases: (i) the last checkpoint before t_i was done during t_j with $j \geq i_0$ or there was no checkpoint before t_i. In this case, $W_i = \tilde{W}_i$; (ii) the last checkpoint before t_i was done during t_j with $j < i_0$ in $\mathcal{S}$, and there was no checkpoint done in $\tilde{\mathcal{S}}$ before t_i. In this case, we have $W_i = R + (t_i - t_j) + \delta_i C$ and $\tilde{W}_i = t_i + \delta_i C$. Since $t_j \leq t_{i_0-1} \leq R$, we get $W_i \geq \tilde{W}_i$. Similarly, we can show that, $\forall i \geq i_0,\ R_i \geq \tilde{R}_i$. Further, since $\mathbb{P}(X > \tilde{a}) \leq \mathbb{P}(X > t_{i_0-1}) \leq 1$, we have
$\mathbb{E}(\tilde{\mathcal{S}}(X)) - \mathbb{E}(\mathcal{S}(X)) \leq \alpha(\tilde{a} - t_1 - C_1) + \beta(\tilde{a} - R_{i_0} - (1-\delta_{i_0-1})T_{i_0-1} - C_{i_0-1}) \cdot$
$\mathbb{P}(X > t_{i_0-1}) \leq (\alpha + \beta)\tilde{a} \leq \varepsilon(\alpha + \beta)\mathbb{E}[X]$.

Finally, $\mathbb{E}(\mathcal{S}(X)) \geq (\alpha+\beta)\mathbb{E}[X] + \gamma$, because this is the cost of an omniscient strategy that makes a single reservation of exactly the right size for each job. Therefore, we obtain $\mathbb{E}(\tilde{\mathcal{S}}(X)) - \mathbb{E}(\mathcal{S}(X)) \leq \varepsilon \cdot \mathbb{E}(\mathcal{S}(X))$. □

Theorem 3. *Given a continuous random variable X on the domain $[a, b]$, where $0 \leq a < b$, and given a constant $\varepsilon > 0$,* DYN-PROG-COUNT(X, ε) *is a $1 + \varepsilon$-approximation algorithm for* STOCHASTIC *and executes in time $\mathcal{O}\big(\frac{1}{\varepsilon^3}\big)$.*

Proof. Given a continuous random variable X of support $[a, b]$, we define the discrete random variable $Y_n \sim (v_i, f_i)_{i=1\ldots n}$ as stated in Equation (9) of Algo-

rithm 1: Let $\mathcal{S}^{\text{opt}} = \{(\tilde{t}_i^o, \tilde{\delta}_i^o)\}_{1 \le i \le |\mathcal{S}^{\text{opt}}|}$ denote the optimal solution for X, and let $\mathcal{S}_n^{\text{dp}}$ the optimal solution for Y_n returned by Theorem 2. We want to show that $\mathbb{E}(\mathcal{S}_n^{\text{dp}}(X)) \le (1+\varepsilon) \cdot \mathbb{E}(\mathcal{S}^{\text{opt}}(X))$. In order to do so, we construct two intermediate strategies $\mathcal{S}_{\varepsilon/3}^{\text{opt}}$ and $\mathcal{S}^{\text{algo}}$ as follows.

First, $\mathcal{S}_{\varepsilon/3}^{\text{opt}} = ((t_i^o, \delta_i^o))_i$ is constructed in such a way that if $\tilde{t}_1^o \ge \min(R, \frac{\varepsilon \mathbb{E}[X]}{3})$, then $\mathcal{S}_{\varepsilon/3}^{\text{opt}} = \mathcal{S}^{\text{opt}}$, otherwise we construct $\mathcal{S}_{\varepsilon/3}^{\text{opt}}$ from $\mathcal{S}^{\text{opt}}$ by Lemma 1 below, with the value $\frac{\varepsilon}{3}$. Then, according to Lemma 1, we have:

$$\mathbb{E}(\mathcal{S}_{\varepsilon/3}^{\text{opt}}(X)) \le \big(1 + \frac{\varepsilon}{3}\big) \cdot \mathbb{E}(\mathcal{S}^{\text{opt}}(X)) \tag{10}$$

Second, $\mathcal{S}^{\text{algo}} = ((t_i^a, \delta_i^a))_{1 \le i \le |\mathcal{S}_{\varepsilon/3}^{\text{opt}}|}$ (hence $|\mathcal{S}^{\text{algo}}| = |\mathcal{S}_{\varepsilon/3}^{\text{opt}}|$), is such that for $1 \le i \le |\mathcal{S}_{\varepsilon/3}^{\text{opt}}|$, we let $(t_i^a, \delta_i^a) = (v_{\pi^o(i)}, \delta_i^o)$. Here, we use the sequence $(v_i)_{i=0\ldots n}$ from Equation (9), and the function π^o defined by $v_{\pi^o(i)-1} < t_i^o \le v_{\pi^o(i)}$. In other words, for each reservation, $\mathcal{S}^{\text{algo}}$ chooses the first discrete value larger than or equal to the corresponding one chosen by $\mathcal{S}_{\varepsilon/3}^{\text{opt}}$, and makes the same checkpointing decision.

Lemma 2. $\mathbb{E}(\mathcal{S}^{\text{algo}}(X)) \le (1 + \frac{\varepsilon}{3}) \cdot \mathbb{E}(\mathcal{S}_{\varepsilon/3}^{\text{opt}}(X))$.

Proof. We use the notations T_i^o, R_i^o, C_i^o, W_i^o for the parameters of $\mathcal{S}_{\varepsilon/3}^{\text{opt}}$, and T_i^a, R_i^a, C_i^a, W_i^a for the parameters of $\mathcal{S}^{\text{algo}}$. From Equation (4), we see that, for $1 \le i \le |\mathcal{S}_{\varepsilon/3}^{\text{opt}}|$, we have $\delta_i^o = \delta_i^a$; $R_i^o = R_i^a$; $C_i^o = C_i^a$; and $W_i^a - W_i^o = T_i^a - T_i^o$. In addition, if $\sigma^o(i)$ (resp. $\sigma^a(i)$) is the index of the last checkpoint before t_i^o (resp. t_i^a), then $\sigma^o(i) = \sigma^a(i)$, and,

$$\begin{aligned}
|T_i^a - T_i^o| &= \left| \left(t_i^a - t_{\sigma^a(i)}^a\right) - \left(t_i^o - t_{\sigma^o(i)}^o\right) \right| \\
&= \left| \left(v_{\pi^o(i)} - v_{\pi^o(\sigma^o(i))}\right) - \left(t_i^o - t_{\sigma^o(i)}^o\right) \right| \\
&= \left| \left(v_{\pi^o(i)} - t_i^o\right) - \left(v_{\pi(\sigma^o(i))} - t_{\sigma^o(i)}^o\right) \right| \\
&\le \max\left(v_{\pi^o(i)} - t_i^o, v_{\pi^o(\sigma^o(i))} - t_{\sigma^o(i)}^o\right) \le \frac{b-a}{n}
\end{aligned}$$

From Equation (7) we have: $\mathbb{E}(\mathcal{S}_{\varepsilon/3}^{\text{opt}}(X)) = \beta\mathbb{E}[X] + \sum_{i=1}^{|\mathcal{S}_{\varepsilon/3}^{\text{opt}}|} \Big(\alpha W_i^o + \beta\big(R_i^o + (1 - \delta_{i-1}^o)T_{i-1}^o + C_{i-1}^o\big) + \gamma\Big) \cdot \mathbb{P}\big(X > t_{i-1}^o\big)$ while $\mathbb{E}(\mathcal{S}^{\text{algo}}(X)) = \beta\mathbb{E}[X] + \sum_{i=1}^{|\mathcal{S}_{\varepsilon/3}^{\text{opt}}|} \Big(\alpha W_i^a + \beta\big(R_i^a + (1 - \delta_{i-1}^a)T_{i-1}^a + C_{i-1}^a\big) + \gamma\Big) \cdot \mathbb{P}\big(X > t_{i-1}^a\big)$.

We first observe that $\mathbb{P}\big(X > t_{i-1}^a\big) \le \mathbb{P}\big(X > t_{i-1}^o\big)$ because $t_{i-1}^a \ge t_{i-1}^o$. We can derive that $\mathbb{E}(\mathcal{S}^{\text{algo}}(X)) - \mathbb{E}(\mathcal{S}_{\varepsilon/3}^{\text{opt}}(X)) \le \sum_{i=1}^{|\mathcal{S}_{\varepsilon/3}^{\text{opt}}|} \Big(\alpha|T_i^a - T_i^o| + \beta(1 - \delta_{i-1}^o)|T_{i-1}^a - T_{i-1}^o|\Big) \cdot \mathbb{P}\big(X > t_{i-1}^o\big) \le \alpha\frac{b-a}{n} + \sum_{i=1}^{|\mathcal{S}_{\varepsilon/3}^{\text{opt}}|-1} \Big(\big(\alpha + \beta(1 - \delta_i^o)\big)\frac{b-a}{n}\Big) \cdot \mathbb{P}(X > t_i^o) \le \frac{b-a}{n}\left(\alpha + (\alpha+\beta)\sum_{i=1}^{|\mathcal{S}_{\varepsilon/3}^{\text{opt}}|-1} \mathbb{P}(X > t_i^o)\right)$.

We also observe that:
$\mathbb{E}(\mathcal{S}^{\text{opt}}_{\varepsilon/3}(X)) \geq \gamma + \sum_{i=1}^{|\mathcal{S}^{\text{opt}}_{\varepsilon/3}|-1} \gamma \cdot \mathbb{P}(X > t_i^o)\,.$

Furthermore, for $1 \leq i \leq |\mathcal{S}^{\text{opt}}_{\varepsilon/3}|$, we have $W_i^o \geq R_i^o + T_i^o \geq \min(R, \tilde{a})$, where $\tilde{a} = \max(a, \min(R, \varepsilon\mathbb{E}[X]/3))$. This is because either $T_i^o \geq \tilde{a}$ according to Lemma 1 (when there was no checkpoint before t_i^o), or $R_i^o = R$ (when there was a checkpoint before t_i^o). Therefore:

$\mathbb{E}(\mathcal{S}^{\text{opt}}_{\varepsilon/3}(X)) \geq \min(\max(a, \varepsilon\mathbb{E}[X]/3), R, C)\Big(\alpha + (\alpha+\beta)\sum_{i=1}^{|\mathcal{S}^{\text{opt}}_{\varepsilon/3}|-1} \mathbb{P}(X > t_i^o)\Big).$

One can observe that:

$$\min(R, \max(a, \min(R, \varepsilon\frac{\mathbb{E}[X]}{3}))) = \min(\max(a, \varepsilon\frac{\mathbb{E}[X]}{3}), R).$$

Using c_0 (line 2, Algorithm 1), we obtain:

$$\mathbb{E}(\mathcal{S}^{\text{algo}}(X)) - \mathbb{E}(\mathcal{S}^{\text{opt}}_{\varepsilon/3}(X)) \leq \frac{c_0}{n} \cdot \mathbb{E}(\mathcal{S}^{\text{opt}}_{\varepsilon/3}(X)) \leq \frac{\varepsilon}{3} \cdot \mathbb{E}(\mathcal{S}^{\text{opt}}_{\varepsilon/3}(X)).$$

□

Lemma 3. $\mathbb{E}(\mathcal{S}^{\text{dp}}_n(X)) \leq \mathbb{E}(\mathcal{S}^{\text{algo}}(X))$

Proof. Given any reservation strategy $\mathcal{S} = \{(t_i, \delta_i)\}_{1 \leq i \leq |\mathcal{S}|}$ such that $\forall i,\ t_i \in \{v_1, \ldots, v_n\}$, we show that:

$$\mathbb{E}(\mathcal{S}(Y_n)) - \mathbb{E}(\mathcal{S}(X)) = \beta\,(\mathbb{E}[Y_n] - \mathbb{E}[X])$$

Indeed, for the two distributions Y_n and X, the only differences in the cost function are: (i) the expectations $\mathbb{E}[Y_n]$ and $\mathbb{E}[X]$; and (ii) the probability values $\mathbb{P}(Y_n > t_i)$ and $\mathbb{P}(X > t_i)\,, \forall i$. But if $t_i \in \{v_1, \ldots, v_n\}$, we have:

$$\begin{aligned}
\mathbb{P}(Y_n > t_i) &= \mathbb{P}(Y_n > v_k) \\
&= \mathbb{P}\big(Y_n \in \cup_{j=k+1}^{n}\{v_j\}\big) = \sum_{j=k+1}^{n} \mathbb{P}(Y_n = v_j) \\
&= \sum_{j=k+1}^{n} \mathbb{P}(X \in]v_{j-1}, v_j]) = \mathbb{P}(X \in]v_k, v_n]) \\
&= \mathbb{P}(X > v_k) = \mathbb{P}(X > t_i)
\end{aligned}$$

We obtain that:

$$\mathbb{E}(\mathcal{S}(Y_n)) - \mathbb{E}(\mathcal{S}(X)) = \beta\,(\mathbb{E}[Y_n] - \mathbb{E}[X])$$

We apply this result to both $\mathcal{S}^{\text{dp}}_n$ and $\mathcal{S}^{\text{algo}}$ and derive that:

$$\mathbb{E}(\mathcal{S}^{\text{dp}}_n(Y_n)) - \mathbb{E}(\mathcal{S}^{\text{dp}}_n(X)) = \mathbb{E}(\mathcal{S}^{\text{algo}}(Y_n)) - \mathbb{E}(\mathcal{S}^{\text{algo}}(X))$$

or equivalently,

$$\mathbb{E}(\mathcal{S}^{\text{dp}}_n(Y_n)) - \mathbb{E}(\mathcal{S}^{\text{algo}}(Y_n)) = \mathbb{E}(\mathcal{S}^{\text{dp}}_n(X)) - \mathbb{E}(\mathcal{S}^{\text{algo}}(X))$$

$\mathcal{S}^{\text{dp}}_n$ is optimal for Y_n, hence $\mathbb{E}(\mathcal{S}^{\text{dp}}_n(Y_n)) - \mathbb{E}(\mathcal{S}^{\text{algo}}(Y_n)) \leq 0$. And finally, the result $\mathbb{E}(\mathcal{S}^{\text{dp}}_n(X)) - \mathbb{E}(\mathcal{S}^{\text{algo}}(X)) \leq 0$. □

Combining Lemma 2, Lemma 3 and Equation (10), we get:

$$\begin{aligned}\mathbb{E}(\mathcal{S}_n^{\mathrm{dp}}(X)) &\leq \mathbb{E}(\mathcal{S}^{\mathrm{algo}}(X)) \leq \left(1+\frac{\varepsilon}{3}\right)\cdot \mathbb{E}(\mathcal{S}_{\varepsilon/3}^{\mathrm{opt}}(X)) \\ &\leq \left(1+\frac{\varepsilon}{3}\right)\left(1+\frac{\varepsilon}{3}\right)\cdot \mathbb{E}(\mathcal{S}^{\mathrm{opt}}(X)) \\ &\leq (1+\varepsilon)\cdot \mathbb{E}(\mathcal{S}^{\mathrm{opt}}(X))\end{aligned}$$

which concludes the proof of Theorem 3. □

3.4 Extensions

All the results presented in Sections 3.1 to 3.3, namely the cost model (Theorem 1), the optimal algorithm for discrete distributions (Theorem 2), and the approximation algorithm for continuous distributions with bounded support (Theorem 3), can be extended to some variants of the problem where the checkpoint strategy is determined a priori.

Indeed, there are two important and natural variants to consider: strategies where no reservation is checkpointed, and strategies where all reservations are checkpointed. The former variant (called No-Ckpt) was studied in our previous work [3], where we derived an optimal algorithm for discrete distributions with reduced time complexity $O(n^2)$ instead of $O(n^3)$ as in Theorem 2. The latter variant (called All-Ckpt) also admits an optimal dynamic programming algorithm of reduced time complexity $O(n^2)$ that can be found in the companion report [9].

4 Performance evaluation

In this section, we evaluate the performance of the different algorithms in simulation. In the following, performance stands for the expected cost of each algorithm under various job execution time distributions, C/R overheads and cost functions. We use jobs that follow a wide range of usual probability distributions as well as a distribution obtained from traces of a real neuroscience application. The code for this section is publicly available on `https://gitlab.inria.fr/vhonore/ckpt-for-stochastic-scheduling`.

4.1 Evaluated algorithms

In addition to the algorithms presented in Section 3, we propose a periodic heuristic for the case of bounded distributions. This strategy, described in Algorithm 2, is a natural policy, where successive reservations differ in length by a constant amount of time T, called the *period*. A checkpoint is performed at the end of each period. Hence, the value of W_i associated with each t_i is constant in this strategy. The algorithm specifies the number of chunks τ in the domain $[a, b]$ of the bounded distribution, thus the period can be computed as

Algorithm 2 ALL-CKPT-PER(X, τ)

1: Let $[a, b]$ be the domain of X, and let $T = \frac{b-a}{\tau}$

2: $(t_i, \delta_i) = \begin{cases} (a + i \cdot T, 1) & \text{for } i = 1, 2, \ldots, \tau - 1 \\ (b, 0) & \text{for } i = \tau \end{cases}$

3: **return** $\mathcal{S}_\tau^{\text{period}} \leftarrow ((t_i, \delta_i))_{1 \le i \le \tau}$

Table 1: Probability distributions and parameter instantiations.

Distribution	**PDF** $f(t)$	**Instantiation**	**Support (in hours)**
Distributions with infinite support			
Exponential (λ)	$\lambda e^{-\lambda t}$	$\lambda = 1.0h^{-1}$	$t \in [0, \infty)$
Weibull(λ, κ)	$\frac{\kappa}{\lambda}\left(\frac{t}{\lambda}\right)^{\kappa-1} e^{-\left(\frac{t}{\lambda}\right)^{\kappa}}$	$\lambda = 1.0h$ $\kappa = 0.5$	$t \in [0, \infty)$
Gamma(α, β)	$\frac{\beta^\alpha}{\Gamma(\alpha)} t^{\alpha-1} e^{-\beta t}$	$\alpha = 2.0$ $\beta = 2.0h^{-1}$	$t \in [0, \infty)$
LogNormal (ν, κ)	$\frac{1}{t\kappa\sqrt{2\pi}} e^{-\frac{(\ln t - \nu)^2}{2\kappa^2}}$	$\nu = 3.0h$ $\kappa = 0.5$	$t \in (0, \infty)$
Pareto(ν, α)	$\frac{\alpha \nu^\alpha}{t^{\alpha+1}}$	$\nu = 1.5h$ $\alpha = 3.0$	$t \in [\nu, \infty)$
Distributions with finite support			
Truncated Normal(ν, κ^2, a, b)	$\frac{1}{\kappa}\sqrt{\frac{2}{\pi}} \cdot \frac{e^{-\frac{1}{2}\left(\frac{t-\nu}{\kappa}\right)^2}}{1 - \text{erf}\left(\frac{a-\nu}{\kappa\sqrt{2}}\right)}$	$\nu = 8.0h$ $\kappa^2 = 2.0h^2$ $a = 1.0h$ $b = 20.0h$	$t \in [a, b]$
Uniform(a, b)	$\frac{1}{b-a}$	$a = 1.0h$ $b = 20.0h$	$t \in [a, b]$
Beta(α, β)	$\frac{t^{\alpha-1} \cdot (1-t)^{\beta-1}}{\text{B}(\alpha, \beta)}$	$\alpha = 2.0$ $\beta = 2.0$	$t \in [0, 1]$
Bounded Pareto(L, H, α)	$\frac{\alpha L^\alpha t^{-\alpha-1}}{1 - \left(\frac{L}{H}\right)^\alpha}$	$L = 1.0h$ $H = 20.0h$ $\alpha = 2.1$	$t \in [L, H]$

$T = \frac{b-a}{\tau}$. Note that for this policy, one can derive optimal strategies for some distributions (such as uniform distributions [9]).

Overall, we evaluate five different algorithms from the following two sets of strategies:

• DYN-PROG-COUNT: This set includes Algorithm 1, and its ALL-CKPT and NO-CKPT variants described in Section 3.4.

• ALL-CKPT-PER: This set includes Algorithm 2, and its NO-CKPT-PER counterpart where checkpointing is not allowed (i.e., $\delta_i = 0, \forall i$).

4.2 Evaluation methodology

We evaluate the performance using two scenarios, both based on the *Reserved Instance* pricing scheme in AWS [2], where the user pays exactly what is requested. In the evaluation, we set $\alpha = 1, \beta = \gamma = 0$:

• *Scenario 1* (Section 4.3): We consider nine usual probability distributions,

five of which have infinite support (Exponential, Weibull, Gamma, LogNormal, Pareto) and four have finite support (Truncated Normal, Uniform, Beta, Bounded Pareto). Table 1 lists all distributions used in simulation with the instantiations of their parameters for evaluation. The first five distributions are truncated and fed as input to Algorithm 1. To do so, we set the upper bound of the infinite support to $b = Q(1 - \upsilon)$, where $Q(x) = \inf\{t|F(t) \geq x\}$ is the quantile function and υ is a small constant. In our simulation, we set $\upsilon = 10^{-7}$. During the discretization procedure in Algorithm 1, we then normalize the probabilities of all discrete values so that they sum to 1. We set $C = R = 360$ seconds (0.1 hour). This checkpointing cost is extracted from [17] and corresponds to an average checkpointing duration, where an optimistic one is 60 seconds and a pessimistic one is 600 seconds. We further discuss the impact of the checkpointing cost on the performance.

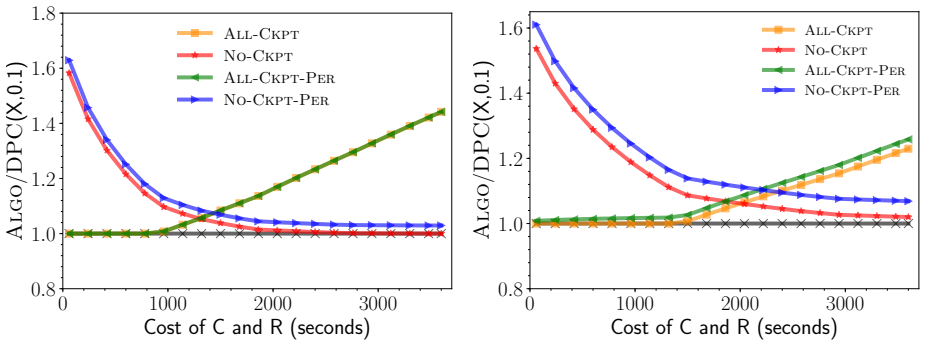
• *Scenario 2* (Section 4.4): In this scenario, we consider execution traces of a real neuroscience application, and fit a LogNormal distribution to its execution times. To further evaluate the robustness of the algorithms, we perturb the parameters of the fitted distribution by varying its mean and standard deviation, and show the impact on the performance.

Additional simulations with different cost models where $\beta \neq 0$ are available in [9], with similar trends as the results presented below.

4.3 Results for Scenario 1

We first evaluate the performance of Dyn-Prog-Count compared to the other strategies, as a function of the values of R and C. Figure 5a presents the performance of these strategies normalized to that of Dyn-Prog-Count (black line for $y = 1.0$) for the Exponential distribution (Figure 5a) and Bounded Pareto distribution (Figure 5b). The results are similar for other distributions [9]. We use $\varepsilon = 0.1$ for Dyn-Prog-Count and its variants. Regarding periodic strategies, we choose the best value for the number of chunks τ in $[1, 1000]$. Not surprisingly, we can observe that when C and R are small, the best result is to use the All-Ckpt strategy while when they are large, one should use the No-Ckpt strategy. There exist thresholds on the sizes of C and R where Dyn-Prog-Count uses a mix of checkpointed and not checkpointed reservations. In that case, the gain of using Dyn-Prog-Count can be up to 10% compared with its variants. An interesting future research direction is to find properties on those thresholds as a function of the probability distribution. Finally, one should observe that the gain achieved with Dyn-Prog-Count compared to the best periodic solution is in general even larger than 10%. The exception is for the Exponential distribution, where one can show [9] that All-Ckpt and its periodic counterpart are identical. This is due to the memoryless property of the Exponential distribution.

We then study the impact of ε on the performance of Dyn-Prog-Count (DPC) when $R = C = 360$s. The idea is that when $\varepsilon = 1$, this theoretically guarantees that the performance is at most twice $(= 1 + \varepsilon)$ that of the optimal, but in practice it can be a lot better. In Figure 6, we study the performance



(a) Exponential (Mean: 3600s) (b) Bounded Pareto (Mean: 6630s)

Figure 5: Expected costs of the different strategies normalized to that of Dyn-Prog-Count$(X, 0.1)$ when $C = R$ vary, for Exponential and Bounded Pareto distributions.

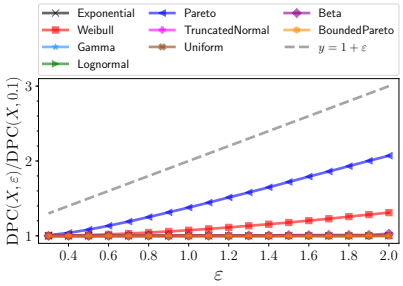


Figure 6: Expected cost of Dyn-Prog-Count(X, ε) as a function of ε for different distributions for X ($C = R = 360$s).

of Dyn-Prog-Count for various values of ε for distributions of Table 1. All performance are normalized by Dyn-Prog-Count for $\varepsilon = 0.1$. We see that in practice, the convergence to the lower bound in performance is fast. Indeed, for $\varepsilon = 1$, almost all distributions already reach convergence, except for Weibull and Pareto (which have a much larger domain of definition and specific properties[1]). For those distributions, we see that they converged for $\varepsilon = 0.1$. For this experiment, the number of chunks n in Dyn-Prog-Count varies between 50 to 1000 depending on the distribution and value of ε, showing the practicality of Dyn-Prog-Count for the target distributions.

Our final evaluation for this scenario is a study of the impact of the period size. So far, we have always chosen the period minimizing the objective functions. Table 2 shows the performance of both variants of the periodic algorithms, All-Ckpt-Per and No-Ckpt-Per, normalized by that of Dyn-Prog-Count

[1]For instance, Pareto is a long-tail distribution, meaning that it has a large number of occurrences that are far from the beginning and central part of its support. Formally, it means that $\frac{1-F(x+y)}{1-F(x)} \to 1$ when $x \to \infty$, $\forall y > 0$.

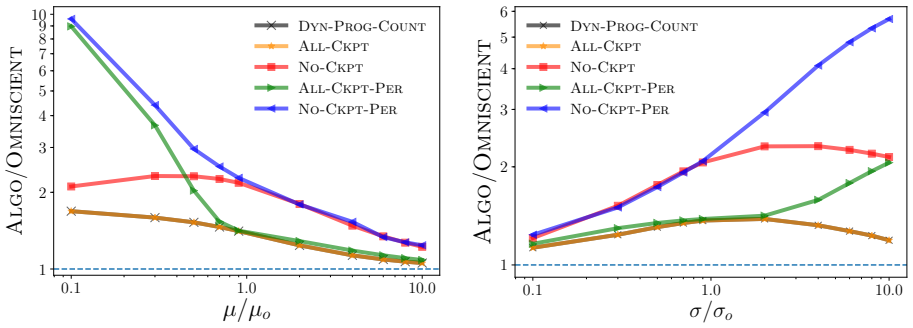
($\varepsilon = 0.1$), when $C = R = 360$s. For each distribution, the second column shows the best period found when τ varies from 1 to 1000 (with its associated cost normalized by that of DYN-PROG-COUNT), and the other columns present results for specific values of τ in that interval. As observed before, ALL-CKPT-PER is in general not able to match DYN-PROG-COUNT (except for some distributions). In addition, we see that NO-CKPT-PER performs even worse than ALL-CKPT-PER. The reason is that the checkpointing cost is relatively low in this setup, so it is preferable to checkpoint more often than never. Finally another observation is that a wrong period size can significantly deteriorate the performance of the periodic algorithms.

Table 2: Expected cost of ALL-CKPT-PER and NO-CKPT-PER, normalized by DYN-PROG-COUNT$(X, 0.1)$. $C = R = 360$s.

Distribution	ALL-CKPT-PER							NO-CKPT-PER						
	Best τ	$\tau = 1$	$\tau = 200$	$\tau = 400$	$\tau = 600$	$\tau = 800$	$\tau = 1000$	Best τ	$\tau = 1$	$\tau = 200$	$\tau = 400$	$\tau = 600$	$\tau = 800$	$\tau = 1000$
Exponential	23 (1.00)	8.60	2.55	4.51	6.48	8.46	10.43	12 (1.38)	8.60	7.81	15.04	22.27	29.50	36.74
Weibull	291 (1.06)	81.56	1.09	1.08	1.17	1.29	1.43	68 (2.54)	81.56	3.85	6.66	9.56	12.49	15.43
Gamma	13 (1.02)	5.35	4.07	7.54	11.02	14.49	17.97	8 (1.26)	5.35	10.44	20.29	30.14	39.99	49.84
LogNormal	9 (1.11)	3.05	4.52	8.24	11.96	15.69	19.41	3 (1.24)	3.05	18.26	35.74	53.21	70.68	88.16
Pareto	574 (1.00)	105.79	1.19	1.02	1.00	1.02	1.04	261 (1.32)	105.79	1.35	1.39	1.57	1.79	2.01
TruncatedNormal	9 (1.10)	2.18	3.28	5.67	8.07	10.46	12.86	2 (1.23)	2.18	30.78	60.69	90.60	120.50	150.41
Uniform	8 (1.01)	1.57	3.17	5.51	7.86	10.20	12.54	1 (1.57)	1.57	51.08	101.33	151.58	201.83	252.09
Beta	2 (1.06)	1.11	30.77	60.99	91.21	121.42	151.64	1 (1.11)	1.11	40.85	81.14	121.42	161.71	202.00
BoundedPareto	32 (1.01)	7.53	1.73	2.71	3.70	4.70	5.69	14 (1.44)	7.53	6.51	12.28	18.06	23.83	29.61

4.4 Results for Scenario 2

We now present the simulation results for a probability distribution fitted to the execution times from the traces of a real neuroscience application (*a code for structural identification of orbital anatomy*) extracted from the Vanderbilt's medical imaging database [15]. Figure 1 shows the execution traces of the application and its fitted LogNormal distribution. Figure 7 presents the performance of different algorithms for this fitted distribution. To evaluate the robustness of algorithms, we also vary the original mean μ_o (Figure 7a) or standard deviation σ_o (Figure 7b) of the distribution from their original values. For readability, all axis are in logscale. We fix the checkpointing cost to $C = R = 600$ seconds and $\varepsilon = 1.0$. For periodic strategies, we use similar brute-force procedure as Scenario 1 to find the period that performs best. The expected costs of the algorithms are normalized by that of an *omniscient* scheduler (blue dashed line), which knows the execution time t of a job *a priori*, and thus would pay the minimum possible cost by making a single reservation of length $t_1 = t$. We observe that DYN-PROG-COUNT always gives the best performance. As previously seen, the checkpointing cost influences the performance of NO-CKPT and ALL-CKPT with regard to DYN-PROG-COUNT. In this setup, since $C = R = 600$ seconds is a value low enough to allow for checkpointing all reservations, the performance of DYN-PROG-COUNT and ALL-CKPT are the same and outperforms NO-CKPT by a wide margin. Simulation with other C/R values can be found in [9] and show similar trends. As for the periodic algorithms, ALL-CKPT-PER has better performance than NO-CKPT-PER. However, both algorithms have worse per-



(a) Variation of μ, $\sigma = \sigma_o = 19.7$h

(b) Variation of σ, $\mu = \mu_o = 21.4$h

Figure 7: Normalized performance of algorithms with omniscient scheduler when μ or σ vary. Basis is the LogNormal distribution in Fig. 1 ($\mu_o = 21.4$h, $\sigma_0 = 19.7$h). $C = R = 600$s, $\varepsilon = 1$. Black (Dyn-Prog-Count) and yellow (All-Ckpt) lines overlap.

formance than Dyn-Prog-Count. The results demonstrate the robustness of Dyn-Prog-Count for a practical application with different distribution parameters.

Finally, when the ratio μ/σ is large (either by increasing the mean (μ/μ_o large), or decreasing the standard deviation (σ/σ_0 small)), the solutions converge to the omniscient scheduler. This could be expected, since in this case the variability becomes negligible and the job behaves similarly to a deterministic job.

5 Experiments

In this section, we conduct real experiments on an HPC platform by using three stochastic neuroscience applications. The focus is to study the performance of different reservation and checkpointing strategies when scheduling multiple jobs in a shared HPC execution environment.

5.1 Experimental setup

The chosen neuroscience applications are described in Table 3 along with their execution characteristics, which are extracted from the Vanderbilt's medical imaging database [15]. In particular, the walltime distributions are obtained by fitting traces of execution times, while the checkpointing/restart costs are obtained by analyzing and averaging memory footprints. Note that, for these applications, restart costs (R) differ from checkpointing costs (C) and depend upon the time-steps at which they are taken. We focus on the evaluation of the following two different sets of strategies:

• An HPC-for-neuroscience strategy (called HPC in Section 5.2), which uses the

Table 3: Characteristics of the chosen neuroscience applications.

Application Type	**Walltime distribution**	C	R
Diffusion model fitting (Qball)	Gamma ($k = 1.18$, $\theta = 34$, $[a, b] = [146s, 407s]$)	90s	40s
Diffusion model fitting (SD)	Weibull ($k = 1043811$, $\lambda = 1174322466$, $[a, b] = [46min, 2.3h]$)	25min	10min
Functional connectivity analysis (FCA)	Gamma ($k = 3.6$, $\theta = 72$, $[a, b] = [165s, 1003s]$)	150s	100s

average of the last 5 runs as the initial reservation length and then increases it by 50% for each subsequent reservation. This strategy is currently used by the MASI group [22] at Vanderbilt to handle stochastic neuroscience applications.
• Our proposed Dyn-Prog-Count strategy and its All-Ckpt variant.

We ran the experiments on a 256-thread Intel Processor (Xeon Phi 7230, 1.30GHz) while submitting jobs through the Slurm scheduler. All three neuroscience applications are sequential (i.e., use a single hardware thread) and perform some medical imaging analysis. The variation in execution time is due to the different characteristics of the input data. However, as we do not have access to the raw input images, we used the information in the logs to simulate the characteristics of the input data, thereby forcing a job to run for a certain walltime and saving a specific amount of data for the checkpoints. The platform under study obeys the ReservationOnly cost model, with $\alpha = 1$, $\beta = \gamma = 0$. In each experiment, we submitted 500 total jobs, and recorded the completion time of each of them. We use the average job *stretch* (defined as the ratio between the total execution time of a job and its actual walltime) to show the individual job performance, and use the *utilization* (defined as the ratio between the sum of all job walltimes and the total time required to execute them) to show the performance of the system for the whole job set. By experimenting on a real system, we investigate the robustness of our strategy: 1) when multiple applications of the same type are running concurrently (and read/write times vary due to congestion while accessing I/O and/or due to application interference); 2) when the C/R costs vary depending upon which time-steps get a checkpoint/restart (i.e., different values for different reservations); and 3) when running multiple job types concurrently.

5.2 Experimental results

Figure 8 shows the performance of the three strategies when submitting 500 jobs from each application to the Slurm scheduler. In this experiment, we manually force the C/R costs to be the same (as in Table 3) for each strategy, in order to study the impact of application interference and runtime system's performance variability on our model. The findings are consistent with the simulation results (in Section 4), showing that Dyn-Prog-Count performs better than its All-Ckpt variant in terms of both system utilization and average job stretch using all three applications. Moreover, both algorithms outperform the simple HPC strategy.

Depending on when the checkpoint is being taken, the checkpoint size and

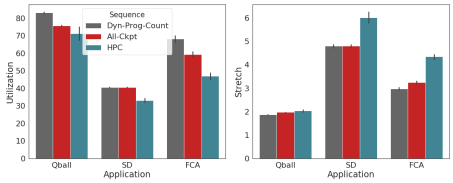


Figure 8: Utilization (higher is better) and average job stretch (lower is better) for Dyn-Prog-Count, All-Ckpt and the HPC strategies.

thus the time to save and restore the application can vary. Figure 9 shows the results when the C/R costs could vary for different reservations. Based on the log traces of these three applications, we noticed that their memory footprints can vary by as much as 30% depending on when the checkpoint is taken (e.g., the checkpoint time can vary between 80 and 110 seconds for Qball). Our experiment generates random checkpoint sizes using a uniform distribution with the mean given by the average checkpoint size from the traces, and forces the application to read/write the corresponding amount at the beginning/end of the execution. In this experiment, we assume that the checkpointing time is included in the request time and is never responsible for applications exceeding their allocated time. While the Dyn-Prog-Count solution is computed using the average C/R costs presented in Table 3, the experimental results show that its performance is robust up to 15-20% variability in the C/R costs. Moreover, the average job stretch appears to be even more stable than the utilization, suggesting that most of the submitted jobs are not impacted by the fluctuation in the C/R costs.

If application-level checkpoint is used, the application is usually aware of the checkpoint size, thus the checkpointing process can start before the reservation is over. The subsequent submissions can easily adapt to this deviation with the first checkpoints that are smaller than the one used to compute the sequence (this is the case for Figure 9). For system-level checkpoint, the application footprint usually remains similar throughout the execution of the application. In case the checkpointing time is causing the application to exceed the reserved time, the submission will fail and subsequent submissions can take this into account by adding the wasted time. The limitation of our method is visible for applications with large variability in checkpointing size, which can be due to multiple factors, either within the application that presents different memory footprints throughout its execution, or by system-level causes, such as I/O congestion or failures. Such large variability in checkpointing size compared to what is used to compute the reservation sequence can result in worse performance when using our method, and the classic HPC model would be preferred in this case. We are currently investigating methods to incorporate variation of checkpointing size into the computation of the optimal reservation sequence, by either using historic information or adapting the subsequent request times based on the sizes of previous checkpoints. We plan to further analyze variable C/R times in the future.

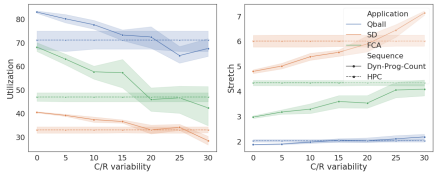


Figure 9: Utilization and average job stretch for each application when varying the C/R costs by different percentages (0 to 30%) using the Dyn-Prog-Count and HPC strategies.

Table 4: Utilization and average job stretch for 10 runs, each using 500 total jobs consisting of a mix of the three applications. The runs are ordered by the best improvement of Dyn-Prog-Count in utilization.

Dyn-Prog-Count		**HPC**		**Improvement**	
Utilization	**Avg Stretch**	**Utilization**	**Avg Stretch**	**Utilization**	**Avg Stretch**
67	2.04	55	2.34	21%	15%
73	1.72	62	2.04	18%	19%
62	2.08	55	2.46	12%	18%
71	1.88	64	2.1	11%	12%
63	2.19	56	2.41	11%	10%
71	1.74	64	1.96	10%	12%
75	1.51	68	1.69	10%	12%
68	2.09	65	2.19	4%	5%
61	2.24	60	2.32	2%	4%
77	1.96	75	1.99	2%	2%

Finally, we conduct experiments in a more realistic scenario by running all three applications simultaneously, and investigating the impact on the different strategies. Specifically, we submitted a total of 500 jobs (100 from Qball, and 200 each from SD and FCA), and kept the C/R costs constant across different reservations to study the sole impact of having several application types executing concurrently. We recorded the utilization and average job stretch when using Dyn-Prog-Count compared to the HPC strategy for 10 different runs choosing different instances from the traces each time. The results are presented in Table 4. We see that Dyn-Prog-Count improves both utilization and average job stretch by 10% on average, and by up to 20% depending on the instances submitted. Overall, these results again illustrate the robustness of our algorithm and confirm its benefit for scheduling stochastic applications on reservation-based platforms, as long as checkpoint costs remain constant for each application.

6 Related Work

We review some related work on reservation-based scheduling and checkpointing in HPC and cloud systems, as well as some prior work on dealing with stochastic applications.

Reservation-based scheduling Batch schedulers are widely adopted by many resource managers in HPC systems, such as Slurm, Torque and Moab. Most batch schedulers use resource reservation in combination with backfilling [23, 25, 27], and rely on users to provide accurate estimates for the walltimes of the submitted jobs. While this works for applications with deterministic resource needs, it can cause resource over-estimation or under-estimation for stochastic jobs with large variations in the walltime, thus degrading system and/or application performance [12, 29].

Clusters of commodity servers that use big-data frameworks such as MapReduce [7] and Dryad [19] offer alternative solutions to running HPC workloads. Schedulers for these frameworks such as YARN [28] and Mesos [16] offer distinct features (e.g., fairness, resource negotiation) to manage the workloads, but they generally also require accurate information regarding the applications' resource demands.

Cloud computing platforms such as Amazon AWS [2] and Google GCP [14] have emerged as another option for executing HPC applications, with a variety of pricing and reservation schemes. Both on-demand and reservation models are available with the latter typically offering a lower price. Several works [1, 5, 8, 31] have studied the pricing strategies for platform providers, as well as delay modeling and cost evaluation for the users.

Stochastic scheduling and checkpointing Many prior works have considered stochastic scheduling for jobs with execution time uncertainty. Most research in this paradigm (e.g., [4,13,21,24,26]) assumes that the execution time of a job follows a known probability distribution and aims to optimize the expected response time or makespan for a set of jobs under various distributions. Most of them, however, do not consider the problem in the context of reservation-based scheduling. In our prior work [3], we have proposed near-optimal reservation strategies for a single job in both HPC and cloud systems. The work was later extended to scheduling a set of stochastic jobs, both sequential and parallel, using backfilling in a reservation-based environment [11, 12].

Another approach to coping with stochastic applications and/or platform unavailability is through checkpoint-restart [18, 30]. To ensure the robustness of the execution, the application's state is periodically checkpointed and in case of interrupt (due to either insufficient reservation or platform failure), the application can be recovered from the last saved checkpoint. In the context of fault tolerance, a lot of work (e.g., [6, 18, 32]) has been devoted to deriving the optimal checkpointing interval that minimizes the checkpointing overhead or resource waste.

In this paper, we present strategies that combine reservation and checkpointing for stochastic jobs with known execution time distributions. To the best of our knowledge, this is the first result to provide performance guarantee on the expected execution time while leveraging checkpointing in reservation-based scheduling environment.

7 Conclusion and Future Work

We have studied the problem of scheduling stochastic jobs running on a reservation-based platform. We presented a model and optimization framework which combine a sequence of reservations with associated checkpointing decisions. We provided an optimal solution via a dynamic programming algorithm in the case of discrete distributions. We also provided approximation scheme for bounded continuous distributions that are arbitrarily close to the optimal. We used both standard distributions and traces from real neuroscience applications to conduct an extensive set of simulations and actual experiments. Altogether, we have demonstrated the effectiveness of these new solutions in comparison with classic strategies. Hopefully, these results should help to convince HPC users and system administrators that significant improvements, in terms of both system and application performance, can be achieved by using a well-chosen reservation sequence rather than a unique reservation of maximum length (the current standard policy).

For future work, we are interested in analytically quantifying the critical checkpointing cost, below (or above) which the best strategy is to always (or never) checkpoint the reservations. This will help to fully characterize the optimal solution for a given application profile. Another interesting direction is to incorporate non-constant checkpointing costs into the optimization problem, in order to design new reservation strategies that will be more robust than our current solutions. This would alleviate the limitation of our approach when checkpointing costs exhibit a large variability.

Acknowledgments We thank the anonymous reviewers for their comments and suggestions. This research was supported in part by the Vanderbilt Institutional Fund. Some of the simulations presented in this paper were carried out using the PlaFRIM experimental testbed, supported by Inria, CNRS (LABRI and IMB), Université de Bordeaux, Bordeaux INP and Conseil Régional d'Aquitaine (see `https://www.plafrim.fr/en/home/`). The remaining simulation resources were provided by the computing facilities MCIA (Mésocentre de Calcul Intensif Aquitain) of the Université de Bordeaux and of the Université de Pau et des Pays de l'Adour.

References

[1] M. Afanasyev and H. Mendelson. Service provider competition: Delay cost structure, segmentation, and cost advantage. *Manufacturing & Service Operations Management*, 12(2):213–235, 2010.

[2] Amazon. AWS pricing information. `https://aws.amazon.com/ec2/pricing/`. Accessed: 2018-10-11.

[3] G. Aupy, A. Gainaru, V. Honoré, P. Raghavan, Y. Robert, and H. Sun. Reservation Strategies for Stochastic Jobs. In *IPDPS*, 2019.

[4] J. Bruno, P. Downey, and G. N. Frederickson. Sequencing tasks with exponential service times to minimize the expected flow time or makespan. *Journal of the ACM*, 28(1):100–113, 1981.

[5] S. Chen, H. Lee, and K. Moinzadeh. Pricing schemes in cloud computing: Utilization-based versus reservation-based. *Production and Operations Management*, 2017.

[6] J. T. Daly. A higher order estimate of the optimum checkpoint interval for restart dumps. *Future Generation Comp. Syst.*, 22(3):303–312, 2006.

[7] J. Dean and S. Ghemawat. MapReduce: Simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113, Jan. 2008.

[8] L. Dierks and S. Seuken. Cloud pricing: the spot market strikes back. In *The Workshop on Economics of Cloud Computing*, 2016.

[9] A. Gainaru, B. Goglin, V. Honoré, G. Pallez, P. Raghavan, Y. Robert, and H. Sun. Reservation and Checkpointing Strategies for Stochastic Jobs (Extended Version). Research Report RR-9294, INRIA, 2019.

[10] A. Gainaru and G. Pallez. Making speculative scheduling robust to incomplete data. In *Scala*, 2019.

[11] A. Gainaru, G. Pallez, H. Sun, and P. Raghavan. Speculative scheduling for stochastic HPC applications. In *ICPP*, 2019.

[12] A. Gainaru, H. Sun, G. Aupy, Y. Huo, B. A. Landman, and P. Raghavan. On-the-fly scheduling versus reservation-based scheduling for unpredictable workflows. *Int. J. High Perf. Computing Applications*, 2019.

[13] A. Goel and P. Indyk. Stochastic load balancing and related problems. In *FOCS*, pages 579–586. ACM, 1999.

[14] Google. GCP pricing information. `https://cloud.google.com/pricing/`. Accessed: 2018-10-16.

[15] R. L. Harrigan, B. C. Yvernault, B. D. Boyd, S. M. Damon, K. D. Gibney, B. N. Conrad, N. S. Phillips, B. P. Rogers, Y. Gao, and B. A. Landman. Vanderbilt university institute of imaging science center for computational imaging XNAT: A multimodal data archive and processing environment. *NeuroImage*, 124:1097–1101, 2016.

[16] B. Hindman, A. Konwinski, M. Zaharia, A. Ghodsi, A. D. Joseph, R. Katz, S. Shenker, and I. Stoica. Mesos: A platform for fine-grained resource sharing in the data center. In *8th USENIX Conf. Networked Systems Design and Implementation*, pages 295–308, 2011.

[17] Z. Hussain, T. Znati, and R. Melhem. Partial redundancy in hpc systems with non-uniform node reliabilities. In *SC*. IEEE Press, 2018.

[18] T. Hérault and Y. Robert, editors. *Fault-Tolerance Techniques for High-Performance Computing*. Springer Verlag, 2015.

[19] M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly. Dryad: Distributed data-parallel programs from sequential building blocks. In *2nd ACM SIGOPS/EuroSys European Conf. Computer Systems*, 2007.

[20] L. Ismail and L. Khan. Implementation and performance evaluation of a scheduling algorithm for divisible load parallel applications in a cloud computing environment. *Software: Practice and Experience*, 45, 2014.

[21] J. Kleinberg, Y. Rabani, and E. Tardos. Allocating bandwidth for bursty connections. In *STOC*, pages 664–673, 1997.

[22] B. Landman. Medical-image Analysis and Statistical Interpretation (MASI) Lab. `https://my.vanderbilt.edu/masi/`.

[23] D. A. Lifka. The ANL/IBM SP Scheduling System. In *JSSPP*, pages 295–303, 1995.

[24] R. H. Möhring, A. S. Schulz, and M. Uetz. Approximation in stochastic scheduling: The power of LP-based priority policies. *Journal of the ACM*, 46(6):924–942, 1999.

[25] A. W. Mu'alem and D. G. Feitelson. Utilization, Predictability, Workloads, and User Runtime Estimates in Scheduling the IBM SP2 with Backfilling. *IEEE Trans. Parallel Distrib. Syst.*, 12(6):529–543, 2001.

[26] J. Niño Mora. Stochastic scheduling. *Encyclopedia of Optimization*, pages 3818–3824, 2009.

[27] J. Skovira, W. Chan, H. Zhou, and D. A. Lifka. The EASY - LoadLeveler API Project. In *JSSPP*, pages 41–47, 1996.

[28] V. K. Vavilapalli, A. C. Murthy, C. Douglas, S. Agarwal, M. Konar, R. Evans, T. Graves, J. Lowe, H. Shah, S. Seth, B. Saha, C. Curino, O. O'Malley, S. Radia, B. Reed, and E. Baldeschwieler. Apache hadoop yarn: Yet another resource negotiator. In *the 4th Annual Symposium on Cloud Computing*, pages 5:1–5:16, 2013.

[29] O. Weidner, M. Atkinson, A. Barker, and R. Filgueira Vicente. Rethinking high performance computing platforms: Challenges, opportunities and recommendations. In *Proceedings of the ACM International Workshop on Data-Intensive Distributed Computing*, pages 19–26, 2016.

[30] K. Wolter, editor. *Stochastic Models for Fault Tolerance, Restart, Rejuvenation, and Checkpointing*. Springer Verlag, 2010.

[31] H. Xu and B. Li. Dynamic cloud pricing for revenue maximization. *IEEE Transactions on Cloud Computing*, 1(2):158–171, July 2013.

[32] J. W. Young. A first order approximation to the optimum checkpoint interval. *Comm. ACM*, 17(9):530–531, 1974.